Lab 9 - Hash Table Experiments
Due (via Bitbucket and hard copy) Wednesday, 13 April 2016
50 points

Lab Goals

- Practice working with Hashtables
- Implement an authentication system
- Answer a few questions to test your knowledge of course material

Assignment Details

In this lab, we will write a small application to experiment with Hashtables. We will write a partial linear probing Hashtable class to store a few data values in an array, and then use that class to store and test whether a user is successfully logging into a system.

Part 1: Hashtable Class (20 points)

To begin, since it has been two weeks since we covered Hashtables, it may be useful to go back to the class notes and review hashtables and the linear probing collision resolution algorithm.

Your Hashtable class will require a backend array and four functions:

1. You need an array to store the data in your hashtable. For simplicity, you can create a keys[] array and a values[] array to separately store each item, such that keys[i] is associated with values[i]. Generics are OK, but you can also assume that all keys and values will be Strings. Each array should be initialized to size = 11. You won’t need to worry about resizing the arrays.

2. A put() function which takes as parameters a (key,value) pair to insert into the hashtable. This function will apply the hash function to the key to determine where in the backend array to store the data, then insert the key and value into that array location. In case of a collision at location i, you should attempt to place the data at location i + 1 and so on until you reach a free location. If you reach position 10 in the array and cannot successfully insert, you’ll have to wrap back around to try location 0.

3. A get() function which takes as a parameter a key and returns the associated value. This function will apply the hash function to the key to determine where in the backend array to locate the value. If the key at that location doesn’t match, continue to check additional
locations in the array until either locating the key or until an empty location is reached (in which case you should return null).

4. A hash() function which takes as a parameter a key and returns an array index (integer). To convert from a String to an integer, let's simply add together the ASCII values of each character in the key. Therefore, the key “mary” will hash to $109 + 97 + 114 + 121 = 441 \mod 11 = 1$, and we would insert “mary” with the associated value into array location 1.

5. A printAll() function which will iterate through the backend arrays and display everything stored inside. Useful for debugging, as well as for proving that your hashtable is working correctly.

Part 2: Authentication System (20 points)

After you have created your Hashtable class, we can start to use it to set up an authentication system. Included in the labs/lab9/data directory is a passwd.txt file formatted with one entry on each line, a username followed by a password. You can treat the username as the key and the password as the value that will be inserted into the hashtable.

To begin, you should open the passwd.txt file, parse through line-by-line, and add each of the (username/key, password/value) pairs into the hashtable.

After all of the items have been entered into the hashtable, you should call the printAll() function to ensure that all data has been stored in the correct location and that all entries have been hashed appropriately. You can use the “mary” example as a sanity check, but you should also compute the hash values of several other entries and ensure that they are being displayed in the correct location.

Now, you should continually prompt the user to enter a username and password. After both data values are retrieved, you should attempt to look up that username and password combination in the hashtable using the get() function. If the password that is returned from get() is null, then the username/password combination doesn’t exist. If the password that is returned from get() is different that the password supplied by the user, then they have entered an incorrect password. If the return value matches the provided input, then the authentication is successful.

You should clearly state something along the lines of “AUTHENTICATION SUCCESS” or “AUTHENTICATION FAILED” after a user attempts to log in. The user should be able to attempt no more than 5 password attempts before the system exits. After an authentication success, the program should also exit.
Part 3: Additional Questions (10 points)

Please answer the following questions thoroughly:

1. In your own words, what is the difference between a checked exception and a runtime exception? Is one better than the other in any way?

2. If we enter 5000 data values into a separate chaining hashtable with 200 array locations, what is the average length of each chain? The smallest possible chain length? The maximum possible chain length?

3. Draw the 13-entry hashtable that results from using the hash function $h(i) = (3i + 5)\%13$ to hash the keys 12, 44, 13, 88, 23, 94, 11, 29, 20, 16, and 5, assuming collisions are handled by separate chaining.

Submission Details

For this assignment, your submission (to both your BitBucket repository and by hardcopy) should include the following:

1. (Upload) Your source code to the requirements in Parts One and Two.

2. (Print and Upload) Sample output showing your authentication system in operation.

3. (Print and Upload) The answers to the questions from Part Three

4. (Print and Upload) An Assignment Information Sheet for your code submission

Before you turn in this assignment, you also must ensure that the course instructor has read access to your BitBucket repository that is named according to the convention cs112s2016-<your user name>.